DAS

Universidade Federal de Santa Catarina

Processamento de Documentos XML em Java

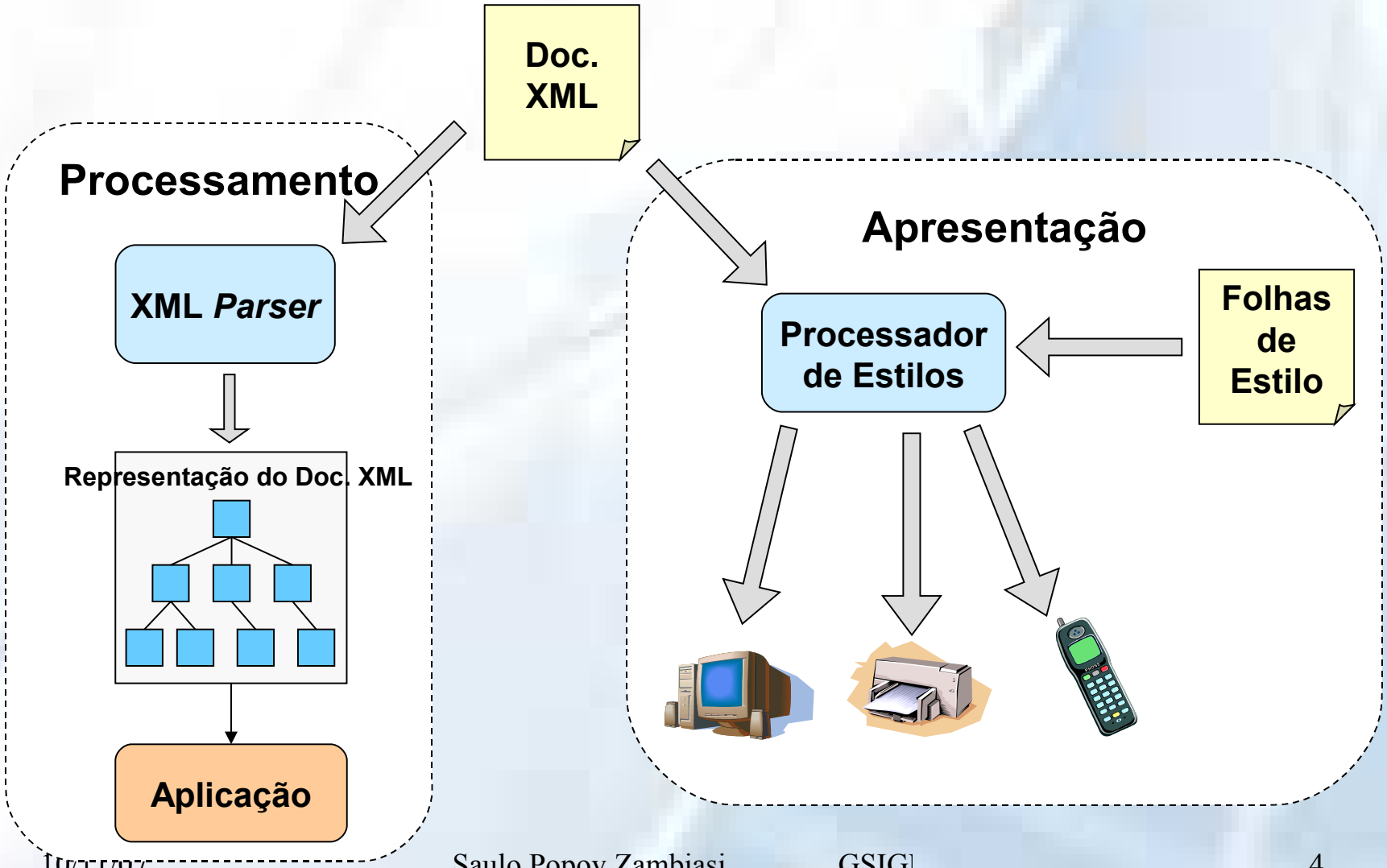
Saulo Popov Zambiasi

Roteiro

- Introdução
- Abordagens básicas: DOM x SAX
- API Java para DOM: JDOM
- Exemplos
- XML usando DAO

Introdução

Processando XML



XML *Parser*

Componente de *software* que analisa documentos XML, gerando uma representação na forma de objetos (**DOM**), ou disparando eventos (**SAX**). Realiza duas operações básicas:

1) *Parsing*: verifica se um documento é *bem-formado*.

2) Validação: verifica se um documento é *válido*.

Abordagens básicas: DOM x SAX

DOM - *Document Object Model*

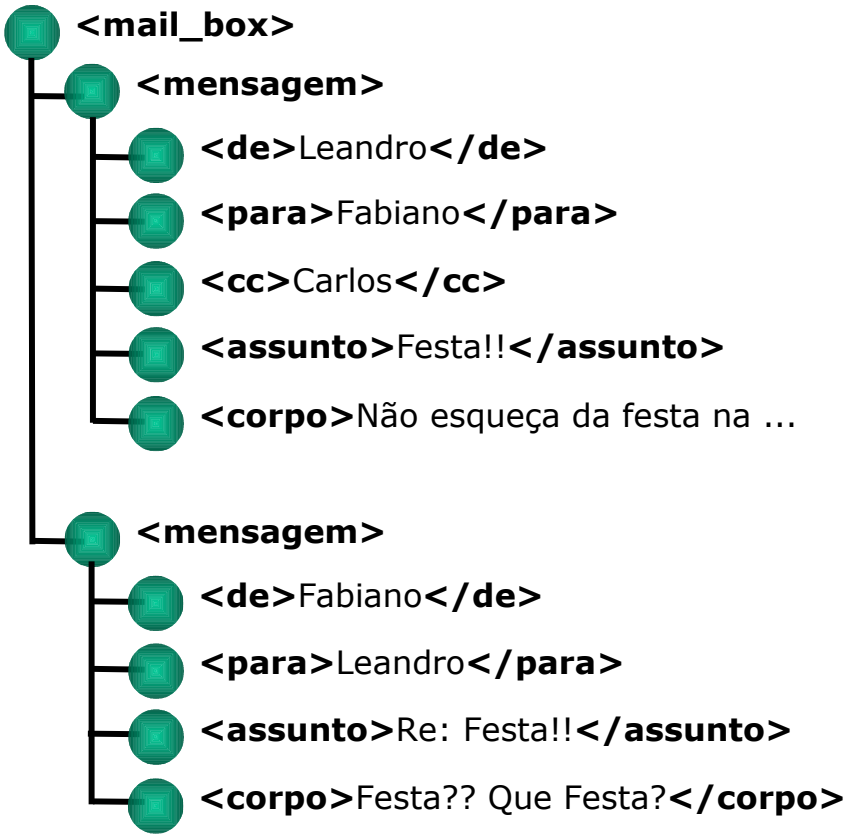
- Modelo de objetos que permite a manipulação de documentos XML;
- Representa o documento XML em forma de **árvore de objetos**;
- Possui uma API que provê acesso/manipulação dos nós da árvore.

DOM - *Document Object Model*

DOCUMENTO XML

```
<mail_box>
  <mensagem data="12/06/2002">
    <de>Leandro</de>
    <para>Fabiano</para>
    <cc>Carlos</cc>
    <assunto>Festa!!</assunto>
    <corpo>Não esqueça da festa na sexta!!</corpo>
  </mensagem>
  <mensagem data="13/06/2002">
    <de>Fabiano</de>
    <para>Leandro</para>
    <assunto>Re: Festa!!</assunto>
    <corpo>Festa?? Que Festa?</corpo>
  </mensagem>
</mail_box>
```

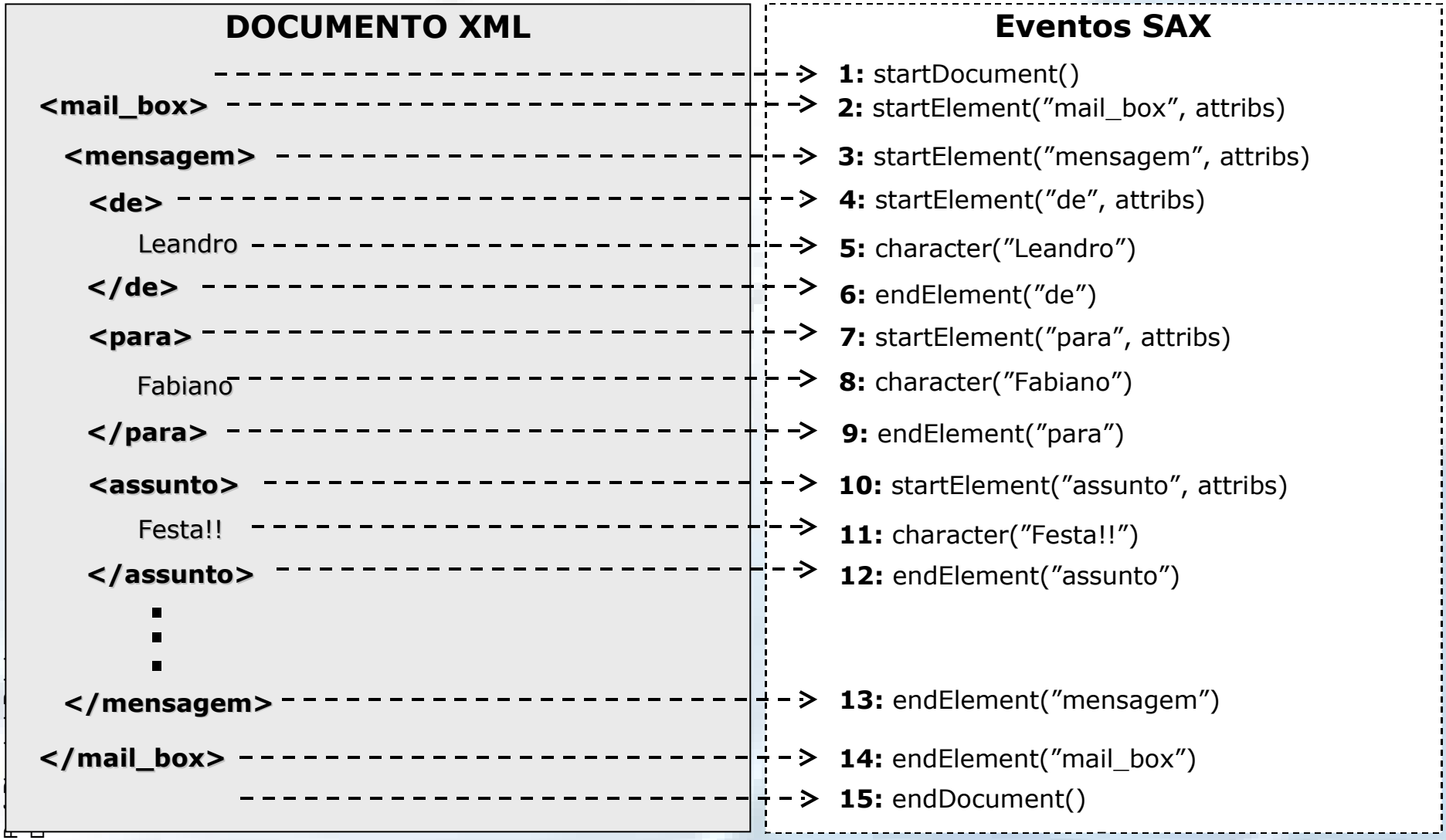
DOCUMENT OBJECT MODEL



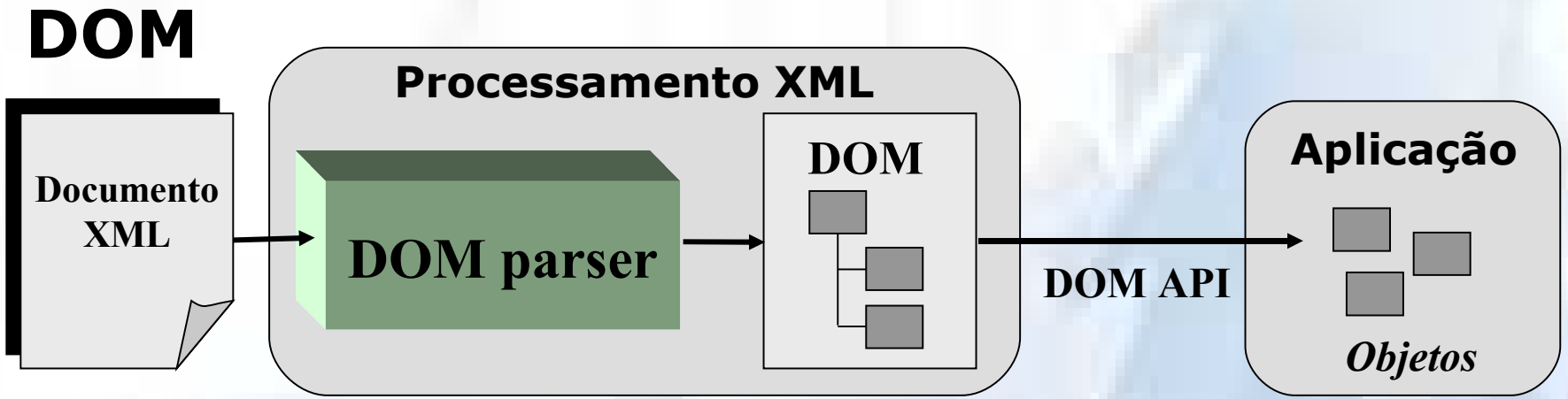
SAX - *Simple API for XML*

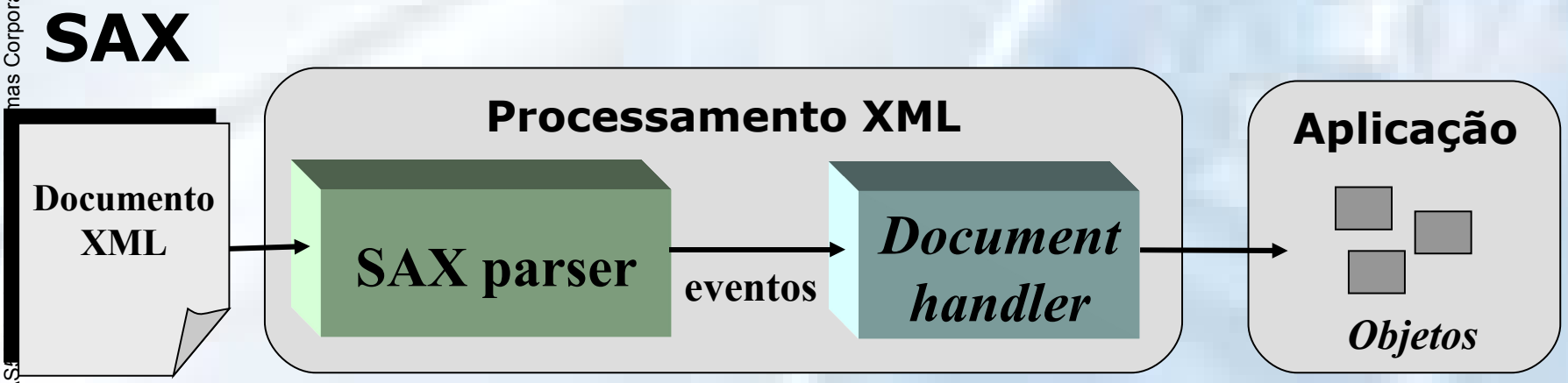
- Acesso ao documento XML através de uma **seqüência de eventos**.
- Como não gera uma representação, é necessária a criação de um modelo de objetos personalizado.
- Necessita também de um objeto responsável por capturar os eventos e salvar as informações no modelo de objetos (*Document Handler*).
- **Os Eventos são disparados quando são encontrados:**
  - *Tag* inicial (que abre um elemento).
  - *Tag* final (que fecha um elemento).
  - Seções de texto (conteúdo dos elementos).
  - Entidades, comentários, instruções de processamento.

SAX - Simple API for XML



DOM x SAX


DOM
Processamento XML
Documento XML
DOM parser
DOM
DOM API
Aplicação
Objetos


SAX
Processamento XML
Documento XML
SAX parser
eventos
Document handler
Aplicação
Objetos

DOM x SAX

SAX

- Possui um desempenho maior, pois não gera uma representação de objetos;
- Aprendizado mais difícil e implementação mais trabalhosa.

DOM

- Esta abordagem é normalmente mais lenta que o SAX, pois gera um modelo de objetos que representa o documento XML;
- Aprendizado fácil e desenvolvimento mais simples.

API Java para DOM: JDOM

JDOM

- Por ter o aprendizado mais fácil, DOM será a abordagem adotada nesta disciplina.
- Ambiente de desenvolvimento Java (JDK) já provê APIs para processamento XML (DOM e SAX) que seguem a recomendação da W3C.
- Entretanto, a API do JDK para DOM apresenta uma certa complexidade que tende a dificultar o seu aprendizado.
- De modo a facilitar o aprendizado, foi adotada uma API mais simples, chamada JDOM (www.jdom.org).
- É uma API simples, mas não implementa as interfaces conforme o padrão da W3C.

JDOM: Principais Classes

- **SAXBuilder:** utiliza um parser *SAX* para gerar um documento JDOM.
- **Document:** documento XML (DOM).
- **Element:** elemento XML.
- **Attribute:** atributo XML. É possível trabalhar com atributos sem utilizar esta classe (apenas com métodos da classe *Element*).
- **DocType:** *Document Type Declaration*.
- **XMLOutputter:** escreve documentos XML em um *stream* de *bytes* (um arquivo, por exemplo).

Processando documentos XML com JDOM

Classe SAXBuilder (Parser XML)

- Representa o parser XML. É usado para validar documentos XML (de arquivos, URLs, *Readers*) e gerar o modelo de objetos (DOM). Lança a exceção *JDOMException*.

Exemplos:

```
// builder1 faz validação (apenas em DTD)
SAXBuilder builder1 = new SAXBuilder(true);

// builder2 faz validação (em XML Schema)
SAXBuilder builder2 = new SAXBuilder(true);
builder2.setFeature("http://apache.org/xml/features/validation/schema", true);

// builder3 não valida, apenas verifica se documento é bem-formado
SAXBuilder builder3 = new SAXBuilder();
```

```
Document doc1 = builder1.build(new File("D:/Temp/pedido_material.xml"));
Document doc2 = builder2.build(new URL("http://xpto.com/pedido.xml"));
```

Obtendo o Elemento Raiz

- O método *getRootElement* da classe *Document* retorna o elemento raiz, representado pela classe *Element*.

```
Element root = doc.getRootElement();
```

- Por exemplo, no processamento do documento abaixo, o método *getRootElement* retorna o elemento *font*.

```
<?xml version="1.0"?>
<font>
  <name>Helvetica</name>
  <size>36</size>
</font>
```

Lendo o conteúdo de um Elemento

- O método *getName* retorna o nome de um elemento. No exemplo anterior, *root.getName()* retorna a *string* "font".
- Para obter os elementos-filhos (apenas do nível seguinte) de um elemento, usa-se o método *getChildren*. Este método retorna uma lista de objetos da classe *Element*.
- O método *getText r*etorna o conteúdo textual de um elemento.

```
Element children = root.getChildren();
for (int i = 0; i < children.size(); i++)
{
   Element child = (Element) children.get(i);
   System.out.println(child.getName());
   System.out.println(child.getText());
}
```

Lendo o conteúdo de um Elemento

- O método *getAttributeValue* retorna o valor de um atributo, dado o seu nome.

```
String value = element.getAttributeValue("id");
```

- Outros métodos:
  - `Element getChild(String name)`
    - Retorna o elemento filho especificado pelo nome.
  - `java.util.List getChildren(String name)`
    - Retorna a lista de elementos filhos (apenas elementos do nível seguinte).
  - `String getChildText(String name)`
    - Retorna o conteúdo textual do elemento filho.

Exemplo 1
Programa que lê um doc. XML

Pedido de Material

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pedidoMaterial SYSTEM "pedido_material.dtd">
<pedidoMaterial>
    <número>P_763</número>
    <códigoFornecedor>P_763</códigoFornecedor>
    <data>2001-09-22</data>
    <dataEntrega>2001-09-23</dataEntrega>
    <item id="1">
      <quantidade>30</quantidade>
      <produto código="XYZ"/>
    </item>
    <item id="2">
      <quantidade>10</quantidade>
      <produto código="ABCD"/>
    </item>
</pedidoMaterial>
```

Exemplo 1: Programa que lê um doc. XML

```
import java.io.File;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

public class XMLProcessor
{

    public static void main(String[] args) throws IOException
    {
        // Processamento do documento XML.
    }

}
```

Exemplo 1: Programa que lê um doc. XML

```
public static void main(String[] args) throws IOException
{
  try
  {
    SAXBuilder builder = new SAXBuilder(true);
    Document doc = builder.build(new File("D:/Temp/pedido_material.xml"));

    Element root = doc.getRootElement();
    System.out.println("Pedido de Material (" + root.getName() + ")");
    System.out.println("-Número: " + root.getChild("número").getText());
    System.out.println("-Código Fornecedor: " + root.getChild("códigoFornecedor").getText());
    System.out.println("-Data: " + root.getChild("data").getText());
    System.out.println("-Data Entrega: " + root.getChild("dataEntrega").getText());
    List items = root.getChildren("item");
    for (Iterator i = items.iterator(); i.hasNext();)
    {
      Element item = (Element) i.next();
      System.out.println("-Item (id: " + item.getAttributeValue("id") + ")");
      System.out.println(" -Código: " + item.getChild("produto").getAttributeValue("código"));
      System.out.println(" -Quantidade: " + item.getChildText("quantidade"));
    }
  }
  catch (JDOMException e)
  {
    e.printStackTrace();
  }
}
```

Exemplo 1: Programa que lê um doc. XML

Execução do programa:

```
Pedido de Material (pedidoMaterial)
-Número: P_763
-Código Fornecedor: F_243
-Data: 2001-09-22
-Data Entrega: 2001-09-23
-Item (id: 1)
 -Código: XYZ
 -Quantidade: 30
-Item (id: 2)
 -Código: ABCD
 -Quantidade: 10
```

Gerando documentos XML com JDOM

Gerando um documento XML

- Uma forma de se escrever documentos XML é construir uma árvore DOM e então gerar o documento.
- Para contruir uma árvore DOM, começa-se com o elemento raiz e depois adiciona-se os seu elementos filhos.
- A cada elemento, adicionar os seus atributos (se for o caso) e seu conteúdo (texto ou outros elementos).
- Vincula-se o elemento raiz a um objeto da classe *Document*, que por sua vez pode ser vinculado a um esquema.
- Finalmente, o texto do documento XML pode ser gerado através da classe *XMLOutputter*.

Gerando um documento XML

- Criando o elemento raiz:

```
Element root = new Element("raiz");
```

- Criando outro elemento, inserindo conteúdo e associando a um atributo:

```
Element child = new Element("filho");
child.setText("123");
child.setAttribute("id", "xpto");
```

- Vinculando o novo elemento ao elemento raiz:

```
root.addContent(child);
```

Gerando um documento XML

- Gerando uma *document type declaration*, para vincular o documento a um DTD:

```
DocType docType = new DocType("raiz", "exemplo.dtd");
```

- Criando um documento, associando o elemento raiz e o *document type declaration*:

```
Document doc = new Document(root, docType);
```

- Gerando o texo do doc. XML com a classe *XMLOutputter*:

```
XMLOutputter outputter = new XMLOutputter
   (Format.getPrettyFormat().setEncoding("ISO-8859-
   1"));
System.out.println(outputter.outputString(doc));
```

Gerando um documento XML

- Resultado:

```
<?xml version="1.0">
<!DOCTYPE raiz SYSTEM "exemplo.dtd">

<raiz>
  <filho id="xpto">123</filho>
</raiz>
```

Gerando um documento XML

- A referência a um XML *Schema* é feita através de um atributo especial (definido em um *namespace* específico) inserido no elemento raiz:

```
Namespace xsi = Namespace.getNamespace("xsi",
   "http://www.w3.org/2001/XMLSchema-instance");
root.setAttribute("noNamespaceSchemaLocation",
   "exemplo.xsd", xsi);
```

```
<?xml version="1.0">

<raiz xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="exemplo.xsd">
  <filho id="xpto">123</filho>
</raiz>
```

Exemplo 2
Programa que gera um doc. XML

Exemplo 2: Programa que gera o doc. XML

```
import java.io.IOException;
import java.io.PrintWriter;

import org.jdom.DocType;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;

public class XMLGenerator
{

    public static void main(String[] args) throws IOException
    {
        // Geração do documento XML.
    }

}
```

Exemplo 2: Programa que gera o doc. XML

```
public static void main(String[] args) throws IOException
{
  Element root = new Element("pedidoMaterial");
  Element elNumero = new Element("número");
  elNumero.setText("P_762");
  root.addContent(elNumero);
  Element elCodigoFornecedor = new Element("códigoFornecedor").setText("F_242");
  root.addContent(elCodigoFornecedor);
  Element elData = new Element("data");
  elData.setText("2001-09-20");
  root.addContent(elData);
  Element elDataEntrega = new Element("dataEntrega").setText("2001-09-21");
  root.addContent(elDataEntrega);
  Element elItem = new Element("item");
  elItem.setAttribute("id", "1");
  root.addContent(elItem);
  Element elQuantidade = new Element("quantidade").setText("20");
  elItem.addContent(elQuantidade);
  Element elProduto = new Element("produto");
  elProduto.setAttribute("código", "XY");
  elItem.addContent(elProduto);
  DocType docType = new DocType("pedidoMaterial", "pedido_material.dtd");
  Document doc = new Document(root, docType);
  XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat().setEncoding("ISO-8859-1"));
  outputter.output(doc, new PrintWriter(System.out));
}
```

Exemplo 2: Programa que gera o doc. XML

Execução do programa:

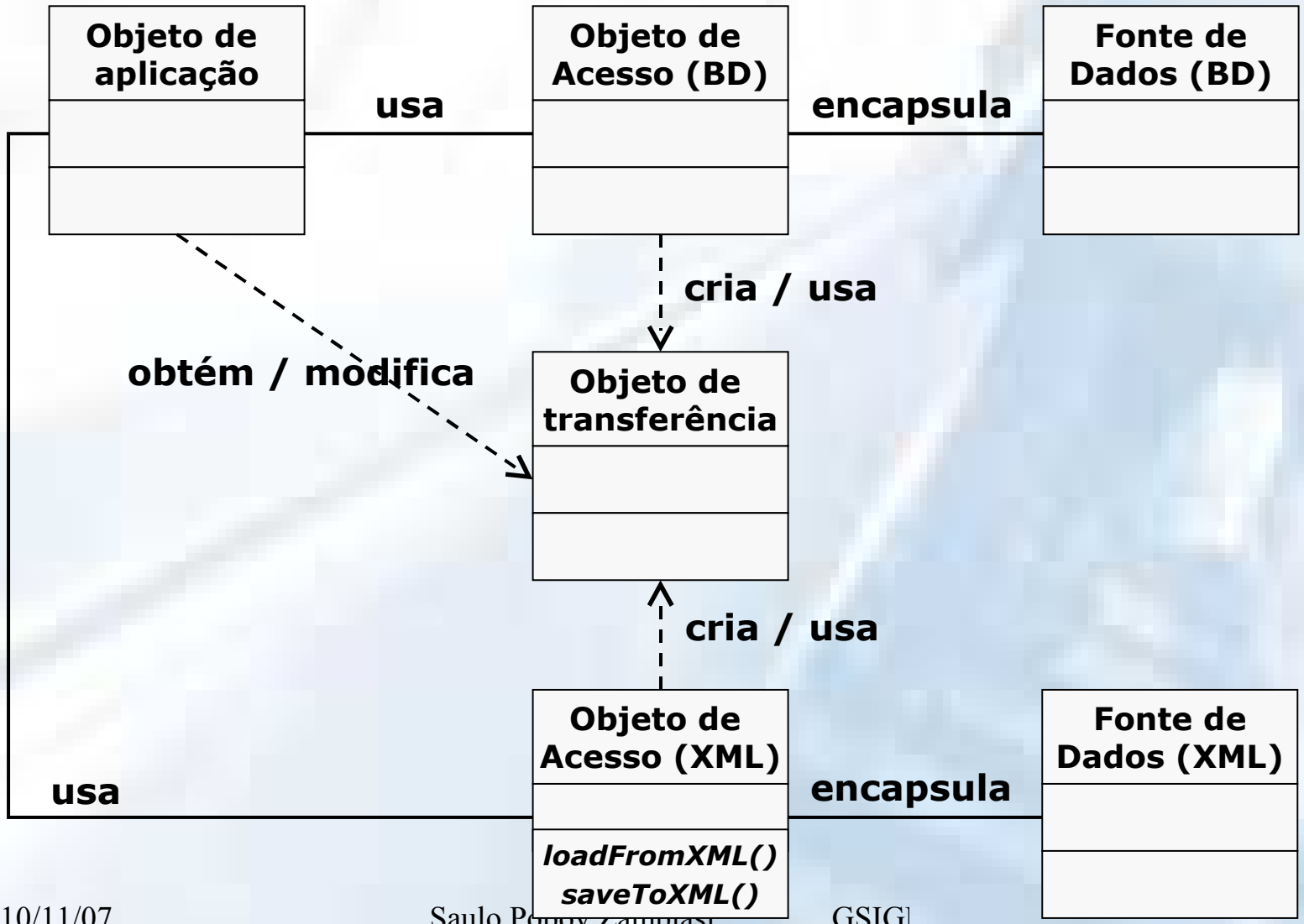
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pedidoMaterial SYSTEM "pedido_material.dtd">

<pedidoMaterial>
  <número>P_762</número>
  <códigoFornecedor>F_242</códigoFornecedor>
  <data>2001-09-20</data>
  <dataEntrega>2001-09-21</dataEntrega>
  <item id="1">
    <quantidade>20</quantidade>
    <produto código="XY" />
  </item>
</pedidoMaterial>
```

XML usando DAO

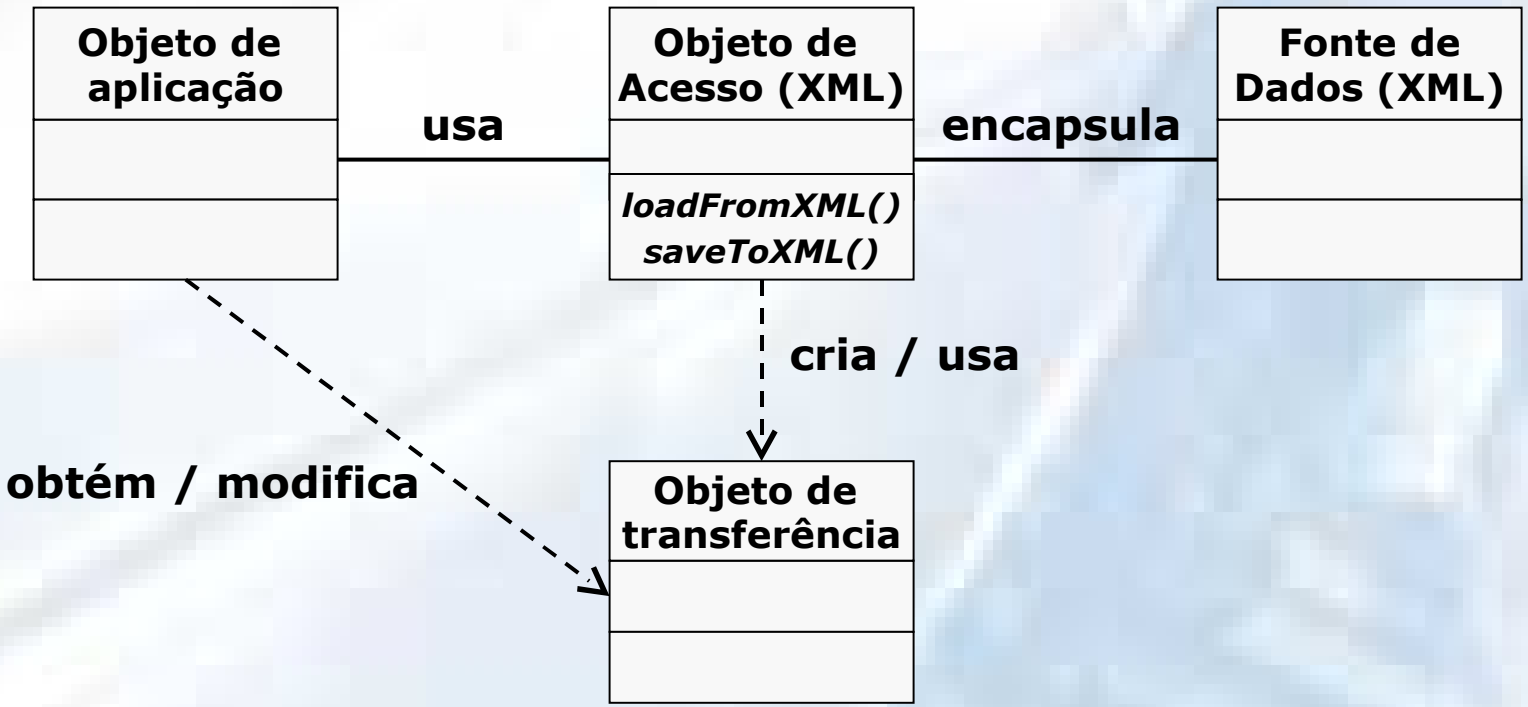
XML usando DAO

- O acesso a documentos XML pode ser abstraído da mesma maneira que uma base de dados.
- Pode-se utilizar o padrão de projeto Java chamando *Data Access Object* (DAO) para o acesso a documentos XML.

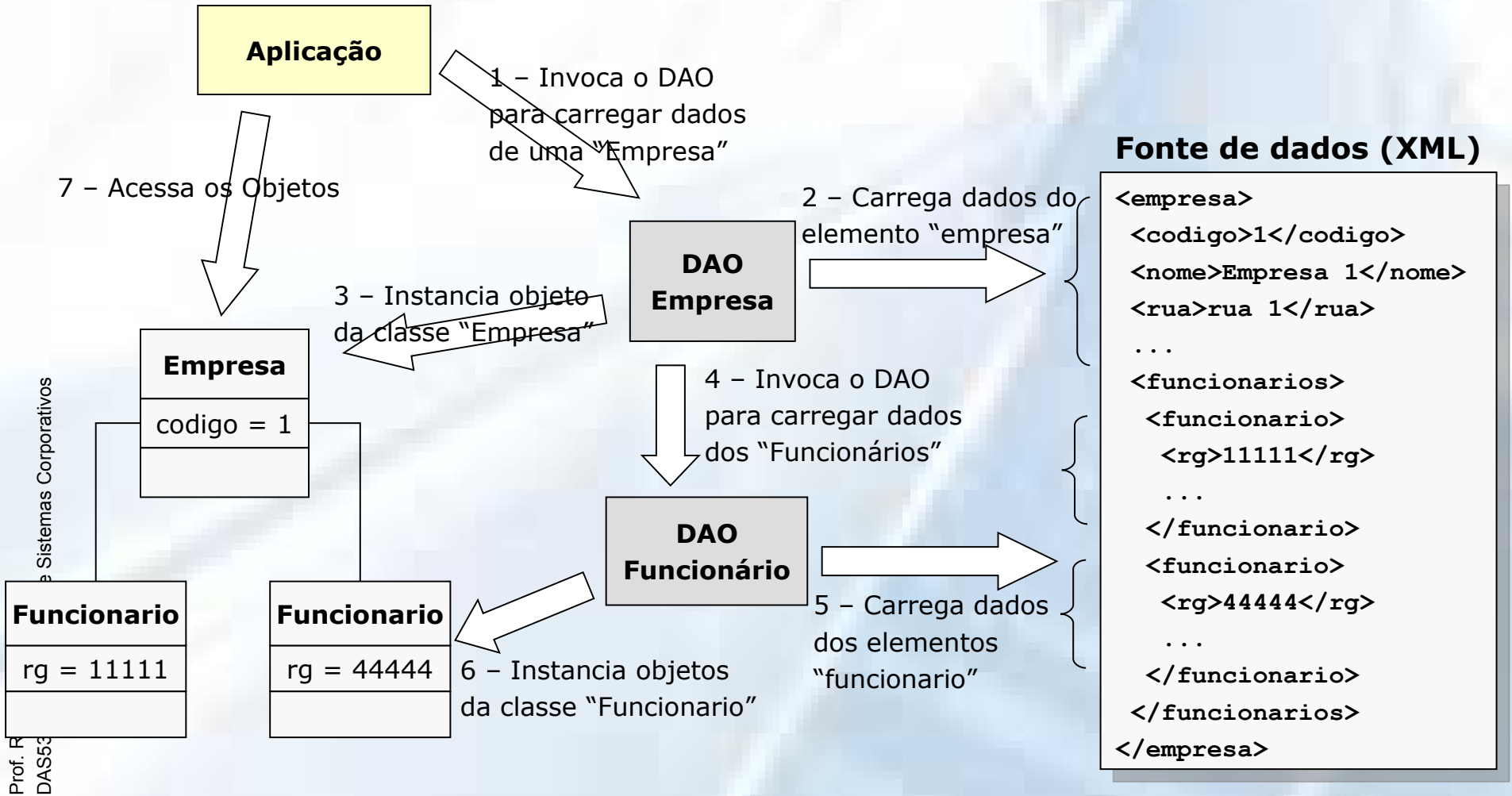
XML usando DAO



XML usando DAO



XML usando DAO: exemplo



Guia (resumido) de Referência do JDOM

JDOM: Principais Classes

- **SAXBuilder:** utiliza um parser *SAX* para gerar um documento JDOM.
- **Document:** documento XML (DOM).
- **Element:** elemento XML.
- **Attribute:** atributo XML. É possível trabalhar com atributos sem utilizar esta classe (apenas com métodos da classe *Element*).
- **DocType:** *Document Type Declaration*.
- **XMLOutputter:** escreve documentos XML em um *stream* de *bytes* (um arquivo, por exemplo).

Classe SAXBuilder

Construtores:

- `public SAXBuilder()`
  - O *parser* criado não faz validação.
- `public SAXBuilder(boolean validate)`
  - O *parser* criado fará validação conforme o parâmetro.

Métodos para geração do DOM:

- `public Document build(java.io.File file) throws JDOMException, java.io.IOException`
  - Gera um documento a partir de um arquivo.
- `public Document build(java.net.URL url) throws JDOMException, java.io.IOException`
  - Gera um documento a partir de um URL.

Classe Document

Construtores:

- `public Document()`
  - Cria um documento vazio.
- `public Document(Element rootElement)`
  - Cria um documento contendo o elemento raiz.
- `public Document(Element rootElement, DocType docType)`
  - Cria um documento contendo o elemento raiz e o *document type declaration* (parâmetros).
- ...

Classe Document

Métodos para leitura:

- `public Element getRootElement()`
  - Retorna o elemento raiz.
- `public DocType getDocType()`
  - Retorna o *document type declaration*.
- ...

Métodos para escrita:

- `public Document setRootElement(Element rootElement)`
  - Define o elemento raiz.
- `public Document setDocType(DocType docType)`
  - Define o *document type declaration*.
- ...

Classe Element

Construtores:

- `public Element(String name)`
  - Cria um elemento com o nome definido pelo parâmetro.
- ...

Métodos para escrita:

- `public Element setText(String text)`
  - Define o texto como conteúdo do elemento.
- `public Element setName(String name)`
  - Define o nome do elemento.
- `public Element setAttribute(String name, String value)`
  - Adiciona um atributo (nome, valor) ao elemento.

Classe Element

Métodos para escrita (continuação):

- `public Element addContent(Content child)`
  - Adiciona um filho (`Content` é superclasse de `Element`)
- ...

Métodos para leitura:

- `public String getName()`
  - Retorna o nome do elemento.
- `public String getText()`
  - Retorna o conteúdo textual do elemento.
- `public String getAttributeValue(String name)`
  - Retorna o valor do atributo especificado.

Classe Element

Métodos para leitura (continuação):

- `public Element getChild(String name)`
  - Retorna o elemento filho especificado pelo nome.
- `public java.util.List getChildren(String name)`
  - Retorna a lista de elementos filhos (apenas elementos do nível seguinte).
- `public String getChildText(String name)`
  - Retorna o conteúdo textual do elemento filho.
- ...

Classe DocType

Construtores

- `public DocType(String elementName, String systemID)`
  - Cria um *document type declaration* contento o nome do elemento raiz e a referência ao DTD.
- ...

Métodos para leitura:

- `public String getElementName()`
  - Retorna o nome do elemento raiz.
- `public String getSystemID()`
  - Retorna a referência ao DTD.
- ...

Classe XMLOutputter

Construtores:

- `public XMLOutputter()`
  - *Outputter* criado escreverá com a formatação padrão, que não define espaçamento nem identação.
- `public XMLOutputter(Format format)`
  - *Outputter* criado escreverá com a formatação definida. A classe *Format* possui um método estático que define um estilo de formatação com espaçamento e identação. Por exemplo:

```
XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
```

  - A classe *Format* também possui um método para definir o *encoding* do documento XML a ser escrito:

```
public Format setEncoding(String encoding)
```

Classe XMLOutputter

Métodos para escrita de documentos XML:

- `public void output(Document doc, java.io.Writer out) throws java.io.IOException`
  - Escreve o documento (`doc`) através do *writer* especificado (Por exemplo, `FileWriter`, `PrintWriter`).
- `public String outputString(Document doc)`
  - Retorna uma *string* contendo o documento XML. O outputter utiliza internamente um *StringWriter*.

Referências

- **W3 Schools – www.w3schools.com**
- **Document Object Model (DOM)**
- **- www.w3.org/DOM/**
- **JDOM API – www.jdom.org**
- **DOM IDL Definitions –www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407/idl-definitions.html**
- **Java Language Binding for DOM Level 3 – www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407/java-binding.html**
- **C++ Language Binding for DOM Level 2 – xml.apache.org/xerces-c/ApacheDOMC++BindingL2.html**